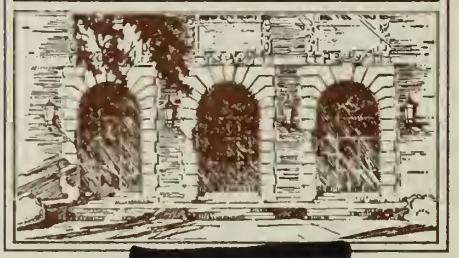

[redacted]

ON COMPUTING THE RANK FUNCTION FOR A SET OF VECTORS

by

Andrew Chi-chih Yao
Foong Frances Yao

February 1975

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-75-699

ON COMPUTING THE RANK FUNCTION FOR A SET OF VECTORS

by

Andrew Chi-chih Yao
Foong Frances Yao

February 1975

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

* This research is supported in part by the National Science Foundation under contract NSF GJ 41538.

ABSTRACT

Let F be a set of n d-component vectors with the componentwise partial order defined on it. It is desired to compute the rank for each vector in F. We show that this can be done in time $O(n(\log n)^{d-1})$ for d = 2, 3, and $O(d\, n^{2 - \frac{1}{d-2}} (\log n)^2)$ for $d \leq \log n$ by employing interesting data structures. When $d \geq n^2 \log n$, an algorithm is given which uses the Four Russians' method to perform efficient partial order verification and runs in $O(d\, n^2/\log n)$ time. Finally, an algorithm is described which, on the average, runs in time $O(d\, n^{1 + \frac{1}{d}} (\log n)^{d-2})$.

1. Introduction

A d-component vector (or d-vector) is a d-tuple of real numbers. The j-th component of a d-vector $\vec{v}$ is written as v_j or $(\vec{v})_j$. We define a partial order $<$ on the set of all d-vectors as follows: $\vec{v} < \vec{w}$ if $\vec{v} \neq \vec{w}$ and $v_j \leq w_j$ for all $j = 1,2,\ldots,d$. For any set F of d-vectors, a rank function r is defined on F where $r(\vec{v})$ is the largest integer k such that there exist k vectors $\vec{w}^{(1)}, \vec{w}^{(2)}, \ldots, \vec{w}^{(k)}$ in F satisfying $\vec{w}^{(1)} < \vec{w}^{(2)} < \cdots < \vec{w}^{(k)} = \vec{v}$.

Recently, Luccio and Preparata[1], Kung[2], and F. Yao[3] have considered the problem of finding the minimal elements for a geven set F of d-vectors, where a minimal element is a vector $\vec{v}$ in F with $r(\vec{v}) = 1$. In this paper, we study the more general problem of determining the rank function for a set of vectors.

Let F be a set of n distinct d-vectors. Computing the rank function r of F in a straightforward manner will take $O(dn^2)$ steps. The purpose of this paper is to present several more sophisticated algorithms which compute r in less time, assuming that n and d are restricted to certain regions. Main results are:

(A) Using worst-case behavior as the cost measure:

(1) For $d = 2,3$, r can be determined in $O(n(\log n)^{d-1})$ steps.

(2) For $d \leq \log n$, r can be determined in $O(dn^{2-\frac{1}{d-2}}(\log n)^2)$ steps.

(3) For $d \geq n^2 \log n$, r can be determined in $O(\frac{dn^2}{\log n})$ steps.

(B) Using average behavior as the cost measure:

For fixed d, r can be determined in $O(n^{1+\frac{1}{d}}(\log n)^{d-2})$ steps on the average if all the $(nd)!$ orderings of the set of numbers $\{ (\vec{v}^{(i)})_j \mid 1 \leq i \leq n,\ 1 \leq j \leq d \}$ are equally probable.

The special case d=2 has also been studied independently by Szymanski [4].

2. Fast Algorithms for $d \leq \log n$

2.1 Introduction

A general procedure to compute the rank function is the following:

1. Arrange the vectors in a sequence $\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(n)}$ such that $\vec{v}^{(j)} \not< \vec{v}^{(i)}$ if $j > i$.
2. Compute the rank function r inductively by

   $r(\vec{v}^{(1)}) = 1$

   $r(\vec{v}^{(j)}) = \max \{ r(\vec{v}^{(i)}) + 1 \mid \vec{v}^{(i)} < \vec{v}^{(j)}, \ i < j \}$ for $j > 1$.

Straightforward implementation of the above prodedure involves $O(dn^2)$ scalar comparisons between components of the vectors. Proper choice of data structures, however, can reduce the running time to $o(dn^2)$ when $d \leq \log n$, as will be seen below.

2.2 The Cases $d = 2$ and 3

In the special cases $d = 2, 3$, the procedure outlined above can be implemented as follows:

1. Arrange the vectors as $\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(n)}$ in increasing lexicographic order.
2. $S_i = \phi$ for $i = 1, 2, \ldots, n$.

   (S_i will be the set of vectors of rank i found so far in the algorithm.)
3. For $j = 1, 2, \ldots, n$, do the following:

   Find (by a binary search) the largest number i_0 in the index set $\{1, 2, \ldots, n\}$ for which there exists a vector $\vec{w} \in S_{i_0}$ with $\vec{w} < \vec{v}^{(j)}$. If i_0 does not exist, set i_0 to 0.

   $r(\vec{v}^{(j)}) = i_0 + 1$

   $S_{i_0+1} = S_{i_0+1} \cup \{\vec{v}^{(j)}\}$

Steps 1 and 2 can be done in $O(n\log n)$ operations. To facilitate the computation in step 3, we shall employ a data structure for each S_i similar to that used in [1]. Let us consider the data structures for $d = 2$ and $d = 3$ separately:

(i) <u>$d = 2$</u> Suppose $r(\vec{v}^{(j)})$ is being computed. Let

$$s_i = \min \{ w_2 \mid \vec{w} = (w_1,w_2) \in S_i \}.$$

It is easy to see that there exists a $\vec{w} \in S_i$ with $\vec{w} < \vec{v}^{(j)}$ iff $s_i \leq v_2^{(j)}$. Therefore, to compute $r(\vec{v}^{(j)})$ for any j takes $O(\log n)$ operations. The total number of operaiions needed in step 3 is thus $O(n\log n)$, since the efforts needed to keep the s_i's updated are small, in fact $O(n)$.

(ii) <u>$d = 3$</u> Again assume $r(\vec{v}^{(j)})$ is being computed. Consider the set T_i of projections of vectors in S_i on the last two components. Let S_i' be the set of minimal elements of T_i, i.e.

$$S_i' = \{(w_2,w_3) \mid \vec{w} \in S_i, \text{ and } (w_2' \; w_3') \not< (w_2,w_3) \text{ for all } \vec{w}' \in S_i\}.$$

Write S_i' as an ordered table $(x_1,y_1), (x_2,y_2), \ldots, (x_\ell,y_\ell)$ with $x_1 < x_2 < \cdots < x_\ell$ and $y_1 > y_2 > \cdots > y_\ell$. To check if $\vec{w} < \vec{v}^{(j)}$ for some $\vec{w} \in S_i$, it is sufficient to find k (by a binary search) such that $x_k < v_2^{(j)} < x_{k+1}$ and see whether $v_3^{(j)} > y_k$. The set S_i' can be structured as an AVL tree, the running time of step 3 is then bounded by

$$n(\log n)^2 + \text{time needed to maintain AVL trees for all } S_i'$$

The second term is of the order $O(n\log n)$, since at most n insertions and n deletions are performed by the algorithm in total.

We have proved the following theorem:

<u>Theorem 2.1</u> Let $d = 2$ or 3. The rank function for a set of n d-component vectors can be computed in time $O(n(\log n)^{d-1})$.

Corollary 2.2 The longest chain in a set of n d-component vectors can be found in time $O(n(\log n)^{d-1})$ for d = 2, 3.

The proof of Corollary 2.2 is immediate.

2.3 General Case $d \leq \log n$

2.3.1 The Algorithm

For $d \leq \log n$ in general, we shall present an algorithm that computes the rank function in time $O(dn^{2-\frac{1}{d+1}})$. Corresponding to the two steps of the general procedure mentioned in Sec. 2.1, this algorithm consists of two phases:

phase 1. Divide the vectors into n/p disjoint sequences $F_1, F_2, \ldots, F_{n/p}$, each containing p vectors. If we write $F_i = (\vec{v}^{(i,1)}, \vec{v}^{(i,2)}, \ldots, \vec{v}^{(i,p)}$ this division satisfies the condition that $\vec{v}^{(i,\ell)} < \vec{v}^{(k,m)}$ implies that (i,ℓ) prededes (k,m) in lexicographic order.

phase 2. $r(\vec{v}^{(k,m)})$ is computed in lexicographic order of (k,m) as follows: For each i, $1 \leq i \leq k$, find

$$f_i = \max \{ r(\vec{v}) \mid \vec{v} < \vec{v}^{(k,m)}, \ \vec{v} \in F_i \}.$$

(If the set on the right hand side is empty, set f_i to 0.)

Let $r(\vec{v}^{(k,m)}) = \max\limits_{1 \leq i \leq k} \{ f_i + 1 \}$.

The division made in phase 1 will be such that, for each i, the set F_i can be characterized by O(d) parameters. This makes it unnecessary, for the computation of f_i in phase 2, to compare $\vec{v}^{(k,m)}$ with all the p vectors in F_i. Thus, instead of $O(dn^2)$, this algorithm will run in time $O(dn^2/p)$, which is $O(dn^{2-\frac{1}{d+1}})$ when p is chosen to be $n^{\frac{1}{d+1}}$.

The following two procedures, partition and compare, describe the data structures to be used for phase 1 and 2 respectively. The algorithm rank is then coded using these procedures.

```
prodedure partition (F, F_1, F_2, ..., F_{n/p});
    comment  A set F of n = p^{d+1} vectors is divided into n/p sequences
             F(i_1,i_2,...,i_p), 1 ≤ i_1,i_2,...,i_d ≤ p.  These sequences
             are then labelled lexicographically in (i_1,i_2,...,i_d)
             as F_1, F_2, ..., F_{n/p}.
    begin  for  j = 0  to  d-1  do
           begin  for each i_1,...,i_j (1 ≤ i_1,...,i_j ≤ p)  do
                  begin perform a stable sort on F(i_1,i_2,...,i_j) by the
                        j+1 st component of the vectors; divide the sorted
                        sequence into p segments of equal lengths
                        {F(i_1,i_2,...,i_j,i_{j+1}) | 1 ≤ i_{j+1} ≤ p} so that
                         v_{j+1} ≤ w_{j+1}  if v ε F(i_1,...,i_j,ℓ), w ε F(i_1,...,i_j,m)
                         and ℓ< m;
                         comment  for j=0, F(i_1,...,i_j) is defined to be F.
                  end
           end
    end
   relabel the sequences {F(i_1,i_2,...,i_d)} as F_1, F_2, ..., F_{n/p}  in
   lexicographic order of (i_1,i_2,...,i_d);
end partition
```

The next procedure <u>compare</u> will be used in implementing phase 2 of the algorithm. Given a number f, $\vec{v}$, and a set of vectors A, it performs the operation

$$f \leftarrow \max \{ f , \{ r(\vec{w}) \mid \vec{w} \in A, \ \vec{w} < \vec{v} \} \} .$$

This procedure acquires efficiency by keeping track of 2d+1 numbers $high_j(A)$, $low_j(A)$, $(1 \leq j \leq d)$ and rmax(A) defined by:

$$high_j(A) = \max_{w \in A} \{ w_j \}$$

$$low_j(A) = \min_{w \in A} \{ w_j \} \qquad 1 \leq j \leq d$$

$$rmax(A) = \max \{ r(\vec{w}) \mid \vec{w} \in A \}$$

```
procedure compare (v⃗, A, f);
    begin  compare v_j with high_j(A) and low_j(A) for j = 1,2,...,d;
            case 1  v_j < low_j(A)  for some j:
                    f retains its original value;
            case 2  v_j ≥ high_j(A)  for all j:
                    f = max { f, rmax(A) };
            case 3  low_j(A) ≤ v_j < high_j(A)  for j = k_1,k_2,...,k_ℓ, and
                            v_j ≥ high_j(A)  for all other j:
                    begin for each w⃗ ε A do
                          begin if v_j ≥ w_j for all j ε {k_1,k_2,...,k_ℓ}
                                then  f = max { f, r(w⃗)};
                          end
                    end
    end
end compare
```

```
algorithm rank(F);
    begin  call partition(F, F_1, F_2, ..., F_{n/p});
           compute high_j(F_i), low_j(F_i) for  1 ≤ j ≤ d,  1 ≤ i ≤ n/p;
           rmax(F_i) = 0  for 1 ≤ i ≤ n/p;
           for  k = 1 to n/p  do
                begin for m = 1 to p  do
                      begin  f = 0;
                             for 1 ≤ i ≤ k-1  call compare (v^(k,m), F_i, f);
                             for  ℓ = 1 to m-1  do
                                  begin if  v_j^(k,m) ≥ v_j^(k,ℓ)  for all 1 ≤ j ≤ d
                                        then  f = max { f, r(v^(k,ℓ))};
                                  end
                      end
                r(v^(k,m)) = f + 1;
                rmax(F_k) = max { rmax(F_k), r(v^(k,m)) }
                end
    end
end rank
```

2.3.2 Analysis of the Algorithm

The cost of algorithm rank can be divided into several parts:

(i) Cost needed to construct $F_1, \ldots, F_{n/p}$: $O(dn\log n)$ operations.

(ii) Cost needed to compute the $\text{high}_j(F_i)$'s and $\text{low}_j(F_i)$'s: $O(dn)$ operations.

(iii) $\sum_{k,m} C^{(k,m)}$ where $C^{(k,m)}$ is the cost for computing $r(\vec{v}^{(k,m)})$.

It is clear that

$$C^{(k,m)} \leq 3dn/p + p\cdot(\text{number of } (i,j)\text{'s for which } \text{low}_j(F_i) \leq v_j^{(k,m)} < \text{high}_j(F_i)) \qquad (1)$$

We will prove in Theorem 2.3 that the parenthesized term on the right hand side of Equ.(1) is bounded by dn/p^2. Therefore,

$$\sum_{k,m} C^{(k,m)} \leq \sum_{k,m} O(dn/p)$$

$$\leq O(dn^2/p).$$

Thus the running time of the algorithm is $O(dn^2/p) = O(dn^{2-\frac{1}{d+1}})$, if we can prove the following proposition.

<u>Theorem 2.3</u> Let x be any number, and j an integer between 1 and d. There are at most n/p^2 sets F_k such that

$$\text{low}_j(F_k) \leq x < \text{high}_j(F_k) \tag{2}$$

<u>Proof</u> For any fixed $i_1, i_2, \ldots, i_{j-1}$, let us consider m, the number of sets F_k such that Equ.(2) is true and $F_k \subset F(i_1, i_2, \ldots, i_{j-1})$ (cf. procedure <u>partition</u>). Now, there is at most one integer ℓ such that

$$\text{low}_j(F(i_1, i_2, \ldots, i_{j-1}, \ell)) \leq x < \text{high}_j(F(i_1, i_2, \ldots, i_{j-1}, \ell))$$

Therefore, if $F_k \subset F(i_1, i_2, \ldots, i_{j-1})$ and Equ.(2) is true, then $F_k \subset F(i_1, i_2, \ldots, i_{j-1}, \ell)$. Since there are at most n/p^{j+1} F_k's in $F(i_1, i_2, \ldots i_{j-1}, \ell)$, we have

$$m \leq n/p^{j+1}.$$

There are p^{j-1} distinct (j-1)-tuples $(i_1, i_2, \ldots, i_{j-1})$. Hence there are a at most $p^{j-1} \cdot (n/p^{j+1}) = n/p^2$ F_k's satisfying Equ.(2). □

We have proved that $O(dn^{2-\frac{1}{d+1}})$ time is sufficient for computing the rank function. The performance of algorithm <u>rank</u> can be further upgraded by the following modifications:

(i) We divide F into sets of size $n^{\frac{1}{d-2}}$ (instead of $n^{\frac{1}{d+1}}$) according to the 2nd, 3rd, ..., d-2nd components of the vectors.

(ii) Initially set all $high_j$'s and low_j's to $-\infty$ and $+\infty$ respectively. The vectors are then input in ascending order of their first components. The $high_j$'s and low_j's are updated each time after a new vector is processed.

(iii) Each set F_i is structured as a set of AVL trees (one for each rank) by the last two components of the vectors as in Sec. 2.2.

These modifications lead to an improvement in the cost of the algorithm as can be easily analyzed. As a result, we have

Theorem 2.4 The rank function of a set of n d-component vectors can be computed in time $O(dn^{2-\frac{1}{d-2}}(\log n)^2)$.

For the longest chain of a set, an analog of Cor. 2.2 is true.

3. The Four Russians' Method and the Case $d \geq n^2 \log n$

For the case $d \geq n^2 \log n$, we shall compute the rank function by first explicitly constructing the partial order matrix for the vectors in F. The procedure we employ to compute the partial order matrix is based on a method for solving the following problem: Given a fixed partial order P, how do we check if an input set of numbers $\{ x_1, x_2, \ldots, x_n \}$ satisfies P? We shall show that this can be done in time $O(n^2/\log n)$ in Sec. 3.1. The complete algorithm to compute rank function is studied in Sec. 3.2.

3.1 Partial Order Verification

Let p be a fixed partial order on a set of n elements. P is given in the form of an $n \times n$ matrix M. For any input set of numbers $S = \{x_1, x_2, \ldots, x_n\}$, it is desired to check whether S satisfies the conditions:

$$x_i \geq x_j \quad \text{if} \quad M_{ij} = 1 \tag{3}$$

We shall prove that, with some preconditioning on the matrix M, this can be done in $O(n^2/\log n)$ time for any input set S.

For a given M, define $\{B_1, B_2, \ldots, B_n\}$, a family of subsets of $\{1, 2, \ldots, n\}$ by

$$j \in B_i \quad \text{iff} \quad M_{ij} = 1 \tag{4}$$

Condition (3) can then be written as

$$x_i \geq \max \{x_j \mid j \in B_i\} \quad i = 1,2,\ldots,n \tag{5}$$

In view of (5), we can perform partial order verification by the following procedure adapted from the Four Russians' Method [5] [6].

(i) (precondition) Partition $\{1, 2, \ldots, n\}$ into $n/\log n$ disjoint segments $I_1, I_2, \ldots, I_{n/\log n}$ each of size $\log n$. Write B_i as $B_i = \bigcup_{j=1}^{n/\log n} C_{ij}$ where $C_{ij} \subset I_j$ for all i, j.

(ii) For each i, compute $m(D) \equiv \max \{x_j \mid j \in D\}$ for all $D \subset I_i$.

(iii) For each i, compute $m(B_i) \equiv \max_j \{m(C_{ij})\}$.

(iv) Test if $x_i \geq m(B_i)$ for all i.

It is not difficult to see that steps (ii), (iii) and (iv) take time $O(n^2/\log n)$. The preconditioning in (i) involves $O(n^2)$ operations. We will use this procedure in the next subsection.

3.2 Computing the Rank Function in $O(dn^2/\log n)$ Time

Let $F = \{\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(n)}\}$ be a set of d-vectors. We define n x n Boolean matrices $M^{(k)}$, $1 \leq k \leq d$, by

$$M_{ij}^{(k)} = 1 \quad \text{iff} \quad \vec{v}^{(i)} \geq \vec{v}^{(j)} \quad \text{for all } \ell = 1,2, \ldots, k. \tag{6}$$

Clearly $M^{(d)}$ properly describes the partial order < defined on F ($M_{ij}^{(d)} = 1$ iff $\vec{v}_i \geq \vec{v}_j$). In the following we will describe a method for finding

$M^{(1)}, M^{(2)}, \ldots, M^{(d)}$ successively. Once $M^{(d)}$ is found, the rank function can then be computed easily.

It is clear that $M^{(k)} = M^{(k-1)}$ iff the following conditions hold:

$$v_k^{(i)} \geq v_k^{(j)} \quad \text{if} \quad M_{ij}^{(k-1)} = 1 \tag{7}$$

Now, according to the result in Sec. 3.1, condition (7) can be checked in $O(n^2/\log n)$ time for the set $\{v_k^{(i)} \mid i = 1,2,\ldots,n\}$ assuming that the preconditioning on $M^{(k-1)}$ has been done. This suggests the following procedure for computing $M^{(d)}$:

```
procedure PO(F, M^(d));
    begin  M^(0) = n x n matrix with all entries equal to 1;
           precondition M^(0) (see Sec. 3.1);
           for  k = 1 to d  do
                begin  if  condition (7) is satisfied                    (8)
                           then  test = true
                           else  test = false;
                       (Use the method in Sec. 3.1)
                       if  test = true
                           then  M^(k) is M^(k-1)
                           else  compute M^(k) and precondition M^(k);   (9)
                                 (Use the fact that M_ij^(k) = 1  iff
                                 M_ij^(k-1) = 1  and  v_k^(i) ≥ v_k^(j).)
                end
    end
end PO
```

Analysis of Procedure PO

(i) Line (8) is executed d times, each taking $O(n^2/\log n)$ operations for a total of $O(dn^2/\log n)$.

(ii) Each time line (9) is executed, it takes $O(n^2)$ operations.

The total running time is therefore

$$T = O(dn^2/\log n + n^2 \cdot L) \tag{10}$$

where L is the number of k's for which $M^{(k)} \neq M^{(k-1)}$. The next theorem shows that $L \leq n(n-1)$, which then implies that

$$T = O(dn^2/\log n + n^4) = O(dn^2/\log n) \quad \text{if } d \geq n^2 \log n.$$

Theorem 3.1 There are at most $n(n-1)$ numbers k for which $M^{(k)} \neq M^{(k-1)}$.

Proof Observe that $M_{ij}^{(k)} = 1$ iff $M_{ij}^{(k-1)} = 1$ and $v_k^{(i)} \geq v_k^{(j)}$. It follows that the number of zero entries in $M^{(k)}$ is strictly greater than that in $M^{(k-1)}$ if $M^{(k)} \neq M^{(k-1)}$. Since $M^{(d)}$ can have at most $n(n-1)$ zero entries, the theorem follows. □

4. An Algorithm With Good Average Behavior

In the previous two sections we have based our algorithms on two different approaches to compute the rank function. Still a third method is the following:

> In the first pass, find the set D_1 of minimal elements in F (using the algorithm of Kung [2] for example). Assign rank 1 to the elements in D_1, and let $F \leftarrow F - D_1$. In the second pass, find the set D_2 of minimal elements in F, and assign them rank 2. The process is repeated until finally $F = \phi$.

We will prove that the preceding procedure runs in time $O(n^{1+\frac{1}{d}}(\log n)^{d-2})$ <u>on the average</u> for any fixed $d \geq 3$. Here we assume that in the input set $F = \{\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(n)}\}$, the dn numbers $F' = \{(\vec{v}^{(i)})_j \mid 1 \leq i \leq n, 1 \leq j \leq d\}$ are randomly distributed, i.e., all the (dn)! orderings for F' are equally probable. It is shown in [2] that finding the minimal elements can be done in $O(n(\log n)^{d-2})$ time. Therefore, we need only show that on the average, no more than $O(n^{\frac{1}{d}})$ passes are necessary for the procedure.

It is easy to see that the number of passes needed on the average is simply the length of the longest chain in the set F on the average. Therefore, it suffices to prove the following theorem:

<u>Theorem 4.1</u> Let $F = \{\vec{v}^{(1)}, \vec{v}^{(2)}, \ldots, \vec{v}^{(n)}\}$ be a set of d-vectors where all the (dn)! linear orderings of the set $F' = \{(\vec{v}^{(i)})_j \mid 1 \leq i \leq n, 1 \leq j \leq d\}$ are equally probable. Then the average length of the longest chain in F is less than $c_d n^{\frac{1}{d}}$ for some constant $c_d > 0$.

<u>Proof</u> Consider the following: Let

$$\pi_i = (p_{i1}, p_{i2}, \ldots, p_{in}) \qquad i = 1, 2, \ldots, d-1$$

be d-1 permutations of (1, 2, ..., n). A <u>common increasing subsequence</u> with respect to the π_i's is a sequence of indices $j_1 < j_2 < \cdots < j_k$ such that

$$p_{ij_1} < p_{ij_2} < \cdots < p_{ij_k} \quad \text{for each } j = 1, 2, \ldots, d-1.$$

Let $L_{d-1}(n)$ be the average length of the longest common increasing subsequence when these d-1 permutations are random.

It is not difficult to show that $L_{d-1}(n)$ is the average length of the longest chain for F. Hence we need only prove that

$$L_{d-1}(n) \leq c_d n^{\frac{1}{d}}. \tag{11}$$

To prove Equ.(11), consider the set of all (d-1)-tuples of permutations $\Gamma = \{(\pi_1, \pi_2, \ldots, \pi_{d-1})\}$. We shall find an upper bound on the number of elements in Γ which have a common increasing subsequence of length $\geq k$. The following is such a bound:

$$C_k^n \cdot (C_k^n)^{d-1} \cdot ((n-k)!)^{d-1}$$

where the first factor C_k^n is the number of possible k positions for a common increasing subsequence $j_1 < j_2 < \cdots < j_k$, the second factor $(C_k^n)^{d-1}$ is the number of possible ways to choose $p_{ij_1}, p_{ij_2}, \ldots, p_{ij_k}$, and the last factor correspond to the different ways of distributing the remaining numbers.

Since $|\Gamma| = (n!)^{d-1}$, the probability P_k of having a common increasing subsequence of length $\geq k$ satisfies

$$P_k \leq \frac{(C_k^n)^d((n-k)!)^{d-1}}{(n!)^{d-1}} = \frac{C_k^n}{(k!)^{d-1}}$$

$$\leq \frac{n^k}{(k!)^d} \simeq \frac{n^k}{(\frac{k}{e})^{dk}(\sqrt{2\pi k})^d}$$

Therefore, for $k \geq (e + \varepsilon)\, n^{\frac{1}{d}}$, we have $P_k \leq \text{constant} \cdot (a_d)^{n^{1/d}}$ where $1 > a_d > 0$. It is easy to derive then that

$$L_{d-1}(n) < (e + 2\varepsilon)n^{\frac{1}{d}} \qquad \text{as} \quad n \to \infty .$$

This proves that $L_{d-1} < c_d n^{\frac{1}{d}}$ for some $c_d > 0$. $\square$

The special case $L_1(n)$ has been studied by Stečkin [7].

REFERENCES

[1] Luccio, F. and Preparata, F. P., On Finding the Maxima of a Set of Vectors, Instituto di Scienze dell'Invormazione, Università di Pisa, Corso Italia 40, 56100 Pisa, Italy, 1973.

[2] Kung, H. T., On the Computational Complexity of Finding the Maxima of A Set of Vectors, 15th Annual Symposium of SWAT, October, 1974.

[3] Yao, F. F., On Finding the Maximal Elements in a Set of Plane Vectors, University of Illinois, Technical Report UIUCDCS-R-74-667, July, 1974.

[4] Szymanski, T. G., A Special Case of the Maximal Common Subsequence Problem, TR #170 Princeton Preprint, January, 1975.

[5] Arlazarov, V. L., Dinic, E. A., Kronrod, M. A. and Faradzev, I. A., On Economical Construction of the Transitive Closure of a Directed Graph, Dokl. Akad. Nauk SSSR 194, 487-448 (in Russian). English translation in Soviet Math. Dokl. 11:5, 1209-1210.

[6] Aho, V. A., Hopcroft, J. E. and Ullman, J. D., The Design and Analysis of Computer Algorithms, Addison Wesley, 1974, 243-247.

[7] Stečkin, B. S., Monotone Sequences in a Permutation of n Natural Numbers, Mat. Zametki 13(1973), 511-514. (English translation in Math. Notes 13(1973), 310-313.)

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-75-699	2.	3. Recipient's Accession No.

4. Title and Subtitle ON COMPUTING THE RANK FUNCTION FOR A SET OF VECTORS	5. Report Date February 1975 6.
7. Author(s) Andrew Chi-chih Yao, Foong Frances Yao	8. Performing Organization Rept. No. UIUCDCS-R-75-699
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, IL 61801	10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF GJ 41538
12. Sponsoring Organization Name and Address National Science Foundation 1800 G Street N.W. Washington, D.C. 20550	13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

Let F be a set of n d-component vectors with the componentwise partial order defined on it. It is desired to compute the rank for each vector in F. We show that this can be done in time $O(n(\log n)^{d-1})$ for d = 2, 3, and $O(d\, n^{2-\frac{1}{d-2}} (\log n)^2)$ for $d \leq \log n$ by employing interesting data structures. When $d \geq n^2 \log n$, an algorithm is given which uses the Four-Russian Method to perform efficient partial order verification and runs in $O(d\, n^2/\log n)$ time. Finally, an algorithm is described which, on the average, runs in time $O(d\, n^{1+\frac{1}{d}} (\log n)^{d-2})$.

17. Key Words and Document Analysis. 17a. Descriptors

rank function

Four Russians' Method

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited	UNCLASSIFIED	18
	20. Security Class (This Page) UNCLASSIFIED	22. Price

FORM NTIS-35 (10-70) USCOMM-DC 40329-P71